

Protecting Content Distribution Networks from Denial of Service Attacks

Kang-Won Lee, Suresh Chari, Anees Shaikh, Sambit Sahu, Pau-Chen Cheng
IBM T. J. Watson Research Center
Hawthorne, NY 10532

Abstract— In this paper, we develop two mechanisms to deter DoS attacks against CDN-hosted Web sites and CDN infrastructure servers. First, we propose a novel request routing algorithm which allows CDN servers to effectively distinguish attacks from legitimate requests. Our scheme, based on a keyed hash function, significantly improves the resilience of servers to DoS attacks. Second, we introduce several site allocation algorithms based on binary codes which insure that an attack on one hosted Web site will have a limited impact on other hosted sites. Our scheme *guarantees* that a specified minimum number of servers remain available for non-victimized sites. Together, the proposed schemes significantly improve the resilience of CDN-hosted Web sites, and complement other work on countering distributed DoS attacks.

I. INTRODUCTION

The problem of detecting and thwarting denial of service (DoS) attacks against Internet servers has drawn considerable attention. These attacks typically flood a network or server with bogus requests, rendering it unavailable to handle legitimate requests. Despite increased awareness about security issues, denial of service attacks remain a challenging problem. DoS attacks often target network resources by generating a large volume of bogus traffic that consumes network bandwidth. The impact of such brute-force attacks can be mitigated, however, by deploying network level mechanisms such as packet filtering and rate limiting [1]. While network level mechanisms provide a first level of protection, they cannot completely prevent attack traffic from reaching its targets. Therefore, considerable attention has been devoted to developing server-side measures to withstand DoS attacks on Internet servers [2], [3].

In today's Internet architecture, many high volume sites are distributed, either by replicating content in several data centers, or via a content distribution service provider (CDSP). Due to the increased server and network capacity available from their geographically distributed infrastructure, CDSPs offer increased resilience to DoS attacks. However, replicating servers or hosting on content distribution networks (CDNs) is not bullet-proof. An attacker can infiltrate a large number of machines using automated tools and launch a large scale distributed DoS (DDoS) attack. Defense against such DDoS attacks is hardest when the attacker uses 'legitimate' packets such as TCP SYN packets, to flood the target site. Hence, we focus on flooding attacks using TCP SYN packets on CDNs. In addition, the shared nature of a CDN's infrastructure can be exploited since an attack on a single CDN-hosted site can affect many other sites hosted by the same CDN. Without a

careful site allocation strategy, the redundancy provided by the CDN offer limited protection.

In this paper, we develop deterrence mechanisms to DoS attacks that are suited for CDN-like environments. Our proposed scheme makes the job of the attacker significantly more difficult by leveraging the request routing system, which directs client requests to the most appropriate CDN server. We also introduce novel site allocation algorithms to provide sufficient isolation among CDN-hosted sites.

As with any Internet security measure, our scheme is not comprehensive by itself. Rather, we propose a set of mechanisms to complement the existing techniques to combat DDoS attacks. Our proposed mechanisms cannot protect, for example, against attackers who target the network bandwidth resources on the links connected to the CDN servers. Such attacks are better addressed by other DDoS countermeasures, such as packet filtering and rate limiting. It should be noted, however, that many observed attacks do not generate enough traffic to consume the bandwidth on today's high-speed links, but are quite sufficient to overwhelm a server [4].

II. RELATED WORK

Network-level mechanisms such as packet filtering [1] and rate-limiting [5], have been designed to mitigate the impact of DDoS attacks. These mechanisms are deployed in network routers to prevent potential attack packets from reaching their destination.

A number of recent studies have proposed techniques to determine the origin of attacks as a deterrence mechanism. Examples of such techniques include "controlled flooding" [6], audit trails [7], [8], and traceback [9], [10]. In packet-based traceback, packets are specially marked as they are forwarded by the routers, and the path back to the origin of can be constructed, given a large enough number of marked packets [9], [10]. While traceback techniques deter potential attackers, they are only good for postmortem analysis.

One of the most common types of attacks on end systems is SYN flooding [11], in which an attacker sends a large number of TCP SYN packets to the target without fully establishing the connection. Numerous defensive measures have been proposed including SYN cookies [3], randomly dropping SYN packets, reducing the time allowed to complete TCP connection establishment. Among these techniques, SYN cookies are most popular in practice because of its simplicity and effectiveness. The basic idea of SYN cookies is to encode the information

about the incoming SYN packets in the sequence number of the corresponding SYN-ACKs. In this way, the server does not have to maintain state for partially established connections, thereby substantially improving the resilience to SYN flood attacks. In Section IV-B, however, we show that SYN cookies alone may not provide enough protection from attacks with very high packet rates as reported in [4].

Unlike most previous work, the main focus of this paper is to develop security mechanisms tailored to CDNs. Recent work by Jung *et al.* on flash crowds in CDNs [12] is closely related, but is different in two ways. First, our scheme differentiates attacks from legitimate requests *on-line*, while Jung *et al.* provide a post-mortem analysis. Second, our scheme improves the resilience of a CDN-hosted site by filtering out attack packets whereas Jung *et al.* proposed a mechanism to dynamically redirect traffic from an overloaded server to less loaded ones.

III. PROBLEM FORMULATION

A. CDN model

We consider a content distribution network model consisting of servers or server clusters distributed in multiple regions. Regions may be arbitrarily defined, though they typically have some topological or geographic significance. Each CDN server in a region is shared in that it serves content of multiple Web sites.

Clients access content from the CDN by first contacting a request router which directs the client to a server within the appropriate region (where the region is chosen based on client proximity, for example). We assume that, within a region, the performance received by the client is equivalent for any server in the region. This assumption is consistent with a recent work by Krishnamurthy et al [13]. Moreover, in certain CDNs, each region may have only single cluster of servers [14], where all servers in the same cluster provide similar performance.

We assume the request router bases its decision on the client IP address, perhaps along with other information about the state of the network or the candidate servers. In practice, the request router may be a specialized DNS server or Web server which chooses a proximal server when the client makes a name resolution request or HTTP request, respectively [15]. If the request router is a DNS server, then additional modifications (e.g., as proposed in [16]) are necessary to expose the client IP address. We note that our proposed mechanisms are designed to operate in each region. In other words, the proposed request routing algorithm selects a target server only from the local region. Similarly, our site allocation algorithm makes allocation decision per region.

B. Assumptions about the attack

In this paper, we assume that the primary target of DDoS attacks on a CDN is the CDN servers. This assumption is based on observations that the attacker can overwhelm a server with a relatively small volume of attack traffic [4], [17]. We also assume that the request routers are less susceptible to DoS attacks than the servers. Unlike Web/application servers, which may perform complex operations, such as DB query

processing, the request routers handle simple request and response operations without having to establish connections or maintain state. Thus, we consider attacks against CDN servers to pose a more immediate threat.

While certain types of DDoS attacks utilize ICMP or UDP packets, recent studies reveal that the majority (more than 90% in the study) of attacks use TCP packets [4]. Also, the observed attack packets commonly has source IP addresses following a random uniform distribution, indicating that source address spoofing is widely used in. Hence, our focus is on flooding attacks using TCP SYN packets with spoofed source IP addresses. By spoofing the source IP address, the attacker will try to hide the true origin of the attack [8], [9] and increase the effectiveness of the attack. Attacks which use genuine IP addresses are not handled by our scheme. Very large-scale attacks using legitimate addresses (e.g., “Code Red”), however, are much more challenging, and countering such attacks is an ongoing research.

C. Quantifying resilience

We begin with a simple notion of *resilience* of a hosting environment. Intuitively, we say server A is more resilient than server B if the attacker must send more attack traffic to bring down server A than to bring down B . Thus, we can define the relative resilience of server A with respect to server B as follows:

Definition 1 (Resilience of a server): Server A is k times more resilient than server B if k times more attack traffic is required to make server A unavailable than to make server B unavailable.

For example, when a site is replicated over a group of n servers then the replicated site provides $O(n)$ resilience compared to a single server because it takes roughly n times more attack traffic to bring down all n servers. We consider CDNs to have n replicated servers and thus have $O(n)$ resilience.¹

Our second metric quantifies the degree of *isolation*, or protection, of a Web site from an attack on another site hosted by the same CDN. For example, consider a CDSP hosting two sites A and B . Ideally, a DDoS attack on site A should not affect the performance or availability of site B , which is true when the two sites are not assigned to any common CDN servers. In practice, though, a single server is shared among multiple sites for resource sharing. Our goal is to maximize the number of servers hosting each site while *guaranteeing* a specified degree of isolation among them. One simple metric for the degree of isolation is the number of servers that are not shared by any two sites:

Definition 2 (Isolation between two sites): Let A and B denote two Web sites, and $S_A = \{s_1, \dots, s_l\}$ and $S_B = \{s'_1, \dots, s'_k\}$ denote the sets of CDN servers allocated to A and B , respectively. We define the *degree of isolation* between A and B to be $\min(|S_A - S_B|, |S_B - S_A|)$. For example, if $S_A = \{s_1, s_2, s_3\}$ and $S_B = \{s_2, s_3, s_4, s_5\}$, the degree

¹Some existing CDN architectures do not provide $O(n)$ resilience, however, since they require the index page of a site to be retrieved from the origin server.

of isolation is 1 because $|S_A - S_B| = |\{s_1\}| = 1$ and $|S_B - S_A| = |\{s_4, s_5\}| = 2$.

In Section V, we show that if sites are assigned to an equal number of servers, then the degree of isolation between two sites is $\frac{d}{2}$ where d (which is always even) denotes the number of disjoint CDN servers.

IV. HASH-BASED REQUEST ROUTING

The key idea of hash-based request routing is to treat requests with legitimate source IP addresses differently from bogus requests with spoofed source IP addresses so that most of the attack packets are preferentially dropped when the CDN is overloaded. In particular, the proposed request routing scheme helps the server to filter out $\frac{n-1}{n}$ fraction of the attack traffic, where n servers are hosting the site in a region.

A. Algorithm description

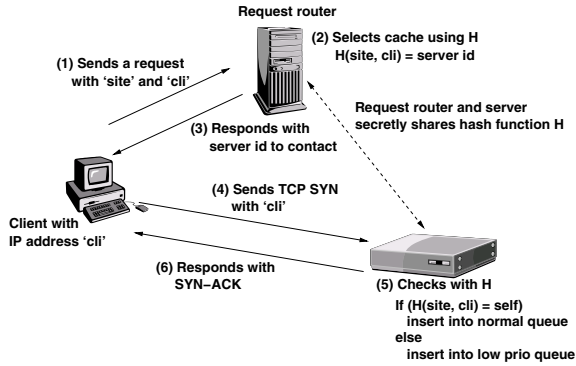


Fig. 1. Operation of hash-based request routing

When a client wants to access a CDN-hosted Web site, it first contacts a request router to find the IP address of the appropriate CDN server to contact. In general, the request routing decision is based on performance or load-balancing metrics. In our approach, however, the hash-based request routing aims to differentiate legitimate requests from potential attack traffic with spoofed IP addresses. This goal is achieved with simple keyed hashing using a secret key shared between request routers and CDN servers [18]. The operation of the hash-based request routing algorithm is described below (see Figure 1).

- 1) The client first sends a request to a request router to find the server address for the target site.
- 2) The request router selects a CDN server based on the target site and the client's IP address using a keyed hash function H , and the secret key K which is shared with the CDN servers. We assume a *simple uniform* keyed hash function $H : \text{IP addr} \rightarrow \text{server id}$. In other words, any given IP address is equally likely to hash into any *server id* in the region.
- 3) The request router responds with the address of the CDN server to contact. Note that the attacker *must* use a legitimate source address to query the request router because it will not receive the response otherwise.
- 4) Upon receiving the response from the request router, the client sends a TCP SYN packet to the server with *server id*. When the server receives the SYN packet it verifies if the source

IP address hashes to its own address using the hash function H and the shared key K .

- 5) If the hash value matches its own address, the CDN server inserts the SYN packet into the "normal" service queue. Otherwise, the SYN packet is inserted into the "low priority" queue. Packets in the normal queue are always served before those in the low priority queue.
- 6) Once the SYN packet is processed by the server, a SYN-ACK packet is returned to the client.

There are a few things to note in this procedure. First, the keyed hash function (Step 2) can dynamically change its behavior by simply changing the key. Second, the attacker cannot discover the mapping for arbitrary IP addresses by permutation because it cannot always receive the responses from the request router (Step 3). Third, the proposed scheme does not generate reverse traffic (TCP SYN-ACK), in response to the attack traffic as SYN cookie does.

Resilience of hash-based request routing: When sending attack traffic, the attacker will try to guess an address that will pass the hashing test at a CDN server. However, the pseudo-randomness of the hash function ensures that the attacker's guess is no better than a random selection.

Suppose the attacker uses random IP addresses. From our assumption of a uniform hash function H , statistically only $\frac{1}{n}$ fraction of the attack traffic, will pass the test at the server, where n is the number of servers in the region. The other $\frac{n-1}{n}$ fraction of the attack traffic will fail the test and be silently dropped. From the attacker's perspective, this scheme requires significantly more attack traffic to bring down a server. In the previous section, we observed that a CDN with n servers has $O(n)$ resilience. With the addition of hash-based request routing, each CDN server will accept only $\frac{1}{n}$ of the attack traffic. This effectively increase the amount of traffic necessary to bring down each server by the factor of n . As a result, an attacker must generate $O(n^2)$ attack traffic in aggregate to victimize all of the CDN servers in the region.

B. Evaluation

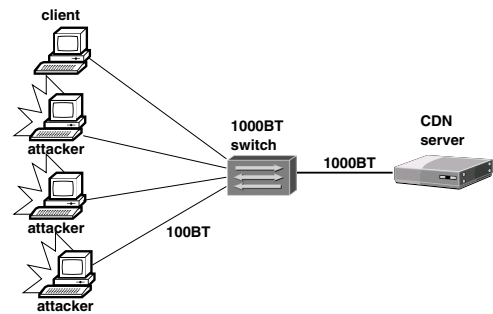


Fig. 2. Testbed configuration

In this section, we present a few performance numbers as a proof of concept. We set up a testbed consisting of one Web server and four clients, all running Linux 2.4.7 on Pentium III 500 MHz PCs (Figure 2). Among the clients, one is assumed to be a legitimate user and the other three serve the role of attackers. The server runs Apache 1.3, and serves local copies

of the CNN.com. In this setting, we examine the performance of a *single* CDN server as if the server is a part of a CDN that employs hash-based request routing. We emulate the number of other servers by controlling the hash parameter. Recall that the proposed hash-based algorithm can filter out $\frac{n-1}{n}$ fraction of attack packets at each server if there are n CDN servers collaborating in the same region.

The attackers generate spoofed TCP packets at a specified rate using raw IP sockets. We use `httperf` [19] to generate request traffic from the legitimate client. For each scenario, five sets of data were collected, where each data consists of results from 1,000 accesses to the Web site. At the server, the Linux SYN cookies implementation is turned on as the basic protection against SYN flood attack. We note that, without SYN cookies, the availability of the server is compromised at a much lower attack rate.

Figure 3 presents the number of timed out requests in the 1,000 accesses from the legitimate user, where timeout is set to 15 seconds. The x-axis represents the attack rate (in packets/sec) on each server. The figure plots the results from the ‘SYN cookies only’ case in comparison with the ‘SYN cookies+hashing’ case, where the number of servers in the region is assumed to be 2, 3, 4, and 5. From the figure, we observe that the proposed hashing scheme reduces timeout events, thereby providing substantially better protection than the case when only SYN cookies used. For example, with the hashing scheme, the legitimate user does not experience timeouts at the attack rate of 3,500 packets/sec, whereas 50% of the requests timeout when only SYN cookies were used. We also observe that the level of protection increases as the number of CDN servers in the region increases.

A similar trend can be found from Figure 4 for the average response time of the HTTP requests that did not time out. As in the previous case, the average response time is significantly lower when the hashing scheme is used than the case without. For the same attack rate, the response time decreases with the number of CDN servers in the same region.

V. ISOLATING THE IMPACT OF THE ATTACK

In this section, we outline strategies to allocate Web sites to different CDN servers in order to isolate the impact of an attack on any individual site. In particular, we want to be able to *guarantee* at least a specified minimum degree of isolation between any two sites in the CDN, while providing good performance. Intuitively, however, allocating a large fraction of the servers to each of two different sites results in significant overlap in the set of servers hosting both of the sites. Thus, an attack which brings down the servers hosting one site also collaterally causes a large loss of service for the other site. Therefore, we have the following two conflicting goals:

- 1) For each Web site, we wish to serve the site from a large number of CDN servers in each region.
- 2) For any pair of Web sites, if one is the target of a DDoS attack then the other should experience minimal service degradation.

The first goal maximizes the throughput of a site hosted by the CDN, and the second protects a Web site from attacks on

any other site.

Our contributions in this paper are to relate the problem of site allocation to a coding theoretic framework and obtain good allocation strategies by adapting carefully chosen codes. We consider the case where there is one level of service, where each Web site is hosted on the same number of CDN servers in each region. In ongoing work we are investigating similar allocation strategies to offer multiple levels of service, for example to allocate more servers to more popular sites.

A. Relating site allocation to codes

Let \mathcal{S} be the set of CDN servers and \mathcal{W} the set of Web sites to allocate to the servers in \mathcal{S} . For each site $w \in \mathcal{W}$ form the bit vector of length $|\mathcal{S}|$ with bit i set if w is allocated to server i . This bit vector is called the *allocation vector* for the site.

Following standard terminology, the *Hamming weight* of a binary vector is defined to be the number of 1s in the vector. The Hamming weight of the allocation vector of a site represents the number of CDN servers that serve content for this Web site. Hence, our goals may be restated as:

- Requirement (1) states that allocation vectors of each site have the largest Hamming weight possible.
- Requirement (2) is that for every pair of sites w_1 and w_2 , the number of servers which serve w_1 but not w_2 (and vice versa) is as large as possible, i.e. if s_1 and s_2 are the allocation vectors of w_1 and w_2 respectively, then the Hamming weights of the bit vectors $(s_1 \wedge \overline{s_2})$ and $(s_2 \wedge \overline{s_1})$ should be as large as possible.²

When all Web sites are treated equally, i.e. when all allocation vectors have equal Hamming weight, we have

$$\begin{aligned} & \text{Hamming weight}(s_1 \oplus s_2) \\ &= 2 \times \text{Hamming weight}(s_1 \wedge \overline{s_2}) \\ &= 2 \times \text{Hamming weight}(s_2 \wedge \overline{s_1}), \end{aligned} \quad (1)$$

where $a \oplus b$ denotes XOR of vectors a and b . The Hamming weight of $(s_1 \oplus s_2)$ is called the *Hamming distance* between s_1 and s_2 . Thus, in this restricted case, our problem is to find allocation vectors with large Hamming weight under the constraint that the Hamming distance between the vectors is as large as possible. Thus the problem of site allocation can be stated as follows:

Allocation problem: Given n , the number of CDN servers in a region, find an efficient algorithm to enumerate a large number of binary vectors each of length n , each vector having Hamming weight exactly h (as large as possible) and the minimum pairwise Hamming distance d between vectors being as large as possible.

Given such an algorithm, we can sequentially generate such n bit vectors and assign them as the allocation vectors for each Web site. Under such an allocation each Web site is served by h CDN servers out of n . If the servers hosting a particular site are all rendered inoperative due to a DDoS attack, then any other site is *guaranteed* to be served by at least $\lfloor \frac{d}{2} \rfloor$ servers. Each Web site thus utilizes $\frac{h}{n}$ of the available capacity and

² \overline{x} denotes the ones-complement of the binary vector x .

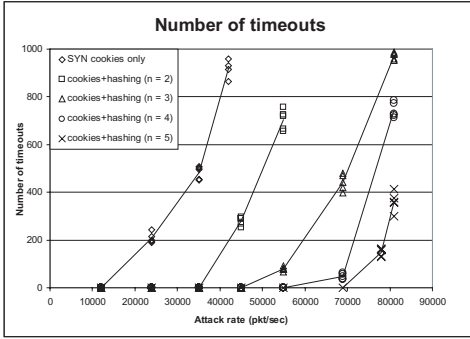


Fig. 3. Number of timeouts vs. attack traffic rate (packets/sec)

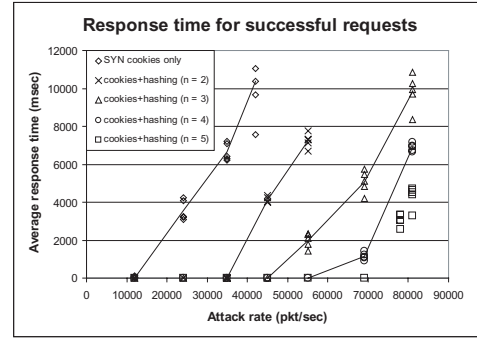


Fig. 4. Response time vs. attack traffic rate (packets/sec)

the resulting loss of service when any one Web site is taken down is at most $(1 - \frac{d}{2h}) * 100$ percent.

B. Allocation strategies from binary codes

In this section, we outline general allocation methods where we try to maximize h and d along with maximizing m , the total number of Web sites which we can accommodate. Our allocation strategies are adaptations of results from coding theory and we outline general constructions without details of actual codes. We refer the interested reader to [20] for comprehensive details about codes and their constructions.

In general, defining codes with many vectors where all codewords have exactly a fixed Hamming weight, is a very difficult problem in coding theory. The few codes that exist to generate constant Hamming weight codewords, generally yield only a small (polynomial in the length of the code) number of codewords. To accommodate a larger number of Web sites, we take arbitrary codes and prune them to yield binary vectors fitting our specification. Our first cut at an allocation strategy is the following naive algorithm:

Algorithm 1: Fix a code of length n with a large minimum distance d . Choose parameter h so that there are enough codewords of Hamming weight h . Then generate all binary vectors with Hamming weight exactly h and output only those vectors which belong to the code.

In this algorithm we first fix a code from a family of codes which fixes the parameter d . Once we fix a code, the distribution on the Hamming weight of the vectors is defined. The parameter h is then chosen to have an allocation for at least m Web sites based on this distribution of Hamming weights.

Besides finding good values for h and d , we also wish to use codes which have explicit constructions and efficient algorithms to enumerate codewords. A particularly good class of codes which have easy algorithms to identify codewords are the class of linear codes. This includes a number of codes such as the Reed–Solomon codes [20].

Definition 3: An (n, k, d) code is a linear code of length n with minimum distance d and the dimension of the linear subspace is k . It is defined by a $k \times n$ binary matrix G called the generator matrix and the set of codewords is obtained by $x \times G$ where x ranges over all binary vectors of length k [20].

Note that linear codes produce codewords with arbitrary Hamming weight. The algorithm to generate codewords is straightforward: Sequentially enumerate vectors of length k and multiply by the generator matrix G . Alternately, a linear code is also defined by its *syndrome matrix* C [20], an $(n - k) \times n$ binary matrix: a n length word x belongs to the code if and only if $x \times C^T = 0$. Using properties of linear codes, our next refinement is the following:

Algorithm 2: Fix an (n, k, d) linear code with a large minimum distance d . Systematically generate all binary vectors of Hamming weight h and retain words x such that $x \times C^T = 0$. Alternately, enumerate vectors y of length k and generate codeword $y \times G$. Retain only those with Hamming weight h .

As before, the parameters d and h are chosen by first fixing a family of linear codes to define d . Once the code is fixed, h is chosen to maximize the number of codewords with Hamming weight h in this code.

We describe another general scheme to obtain allocation vectors, which focuses on a particular value for h . Intuitively, if h is too large, then there are few codewords of Hamming weight h . Also, choosing too large a value for h makes the maximum distance (which can be at most $n - h$) small. On the other hand if h is small, then each Web site is served by at most h CDN servers and thus results in wasted capacity. A particularly good value for h is $\frac{n}{2}$: this is the weight at which we have the maximum number of binary vectors and hence potentially a large number of codewords. For $h = \frac{n}{2}$ we can use the following:

Algorithm 3: Fix a code C of length n with minimum distance d . Define a modified code C' such that for each codeword $c \in C$, C' contains the $2n$ length word $c' = c\bar{c}$.

In the modified code C' , each codeword has length $2n$ and weight exactly n (half the length of the code). The minimum distance between words in C' is at least $2d$. This is a quick way to use any code to produce words of constant Hamming weight with $h = \frac{n}{2}$. The number of codewords in C' is the same as that of C , but now each codeword can be used as an allocation vector.

These algorithms are general methods to convert codes into allocation strategies for Web sites to CDN servers. Plugging good codes into the constructions yields good allocation strategies. The equivalence holds in the other direction: any

allocation strategy can be converted into a code. This equivalence is useful to verify if allocation strategies with certain parameters are possible: There are a number of tables [21] which list (for small values of n), given values for h and the distance d , the maximum number of codewords possible in such a code.

C. Example

Suppose that the CDSP wishes to host 100 Web sites with the guarantee that if a Web site is attacked, all remaining sites have at least 3 functioning servers in the CDN region. Restated, the problem is: *given* $m = 100$ and minimum distance $d = 6$, find optimal values for n and h (See Table I). The first step is to find the minimum value of n for which a code with distance $d = 6$ and at least $m = 100$ codewords exists. From standard tables (see [22]), we see that the minimum possible value for n is 15. Our first cut is to use a very specialized non-linear code [22] which yields about 128 codewords with Hamming weight 8 and with length $n = 16$. This is a fairly optimal allocation strategy using an esoteric non-linear code.

Another allocation can be obtained using the Reed–Solomon code of length $n = 21$ with a distance of $d = 5$ which yields 512 codewords. Inspecting the distribution of the number of codewords for each Hamming weight, we find the number of codewords is maximized at weights 10 and 11. We choose $h = 11$ and select only codewords of weight 11 which yields 126 codewords. For these constant weight words, the distance is actually 6.

A slightly less optimal, but straightforward, allocation is to use Algorithm 3 choosing the code \mathcal{C} to be the Hamming code of length 15 and distance 3. With our parameters the Hamming code has 2048 codewords. Plugging this code into the Algorithm 3 gives us an easily implementable allocation where $n = 30$ and each Web site is assigned to at least 15 servers. While not optimal, the code yields a large number of codewords which gives us the flexibility to expand to more Web sites.

We have chosen these codes from many possibilities, to illustrate the tradeoffs. Optimal codes generally tend to be non-linear with complex encoding algorithms. Straightforward choices for codes such as the Reed–Solomon code give us slightly less optimal values of n .

D. General Allocation Strategies

In this section we discuss a number of possible allocation strategies, using various codes to place different emphasis on the number of hosted Web sites (m), number of CDN servers per hosted site (h), and the degree of isolation between sites (d). Table II summarizes the trade-offs of site allocation using these codes.

Allocations for a small number of Web sites: Our first case is when m is small compared to n . If $m \leq (2n - 2)$ we can use subsets of Hadamard codes [20], [21] and get very good guarantees on the Hamming distance. The Hadamard code is a $(n, \log(2n), \frac{n}{2})$ linear code with $2n$ codewords. In fact, using these codes one can construct $2n - 2$ binary vectors each with

a Hamming weight $\frac{n}{2}$ with minimum pairwise distance $\frac{n}{2}$. With these as allocation vectors, we can assign each site to $\frac{n}{2}$ servers and guarantee that a site will always be served by $\frac{n}{4}$ servers. For small values of m , we can therefore get very good guarantees on resilience.

Codes with efficient algorithms: A good class of codes with a large minimum distance and efficient algorithms are Reed–Solomon codes. Choosing parameters carefully, and using Algorithm 3 stated above, given n , we can use Reed–Solomon codes to enumerate an exponential number ($2^{c_1 n}$) of codewords with a minimum distance of at least $\frac{n}{c_2 \log(n)}$, where c_1 and c_2 are constants. Thus, we can guarantee that no Web site will suffer more than a $\log(n)$ factor drop in service under attack. Although this is high, these codes have the advantage that allocation algorithms are easily implemented.

Allocation strategies for a range of parameters: There are a number of advanced codes which can be converted to good allocation strategies. Care should be taken, however, since they typically have complex algorithms for encoding, and yield the best parameters only for large values of n . One such family of codes are Justesen codes [20]. Plugging these codes in Algorithm 3, gives us an algorithm yielding an exponential number of allocation vectors where each Web site is allocated to $\frac{n}{2}$ servers and we can guarantee that a Web site which is not under attack will at most suffer a small constant factor loss of service.

VI. DISCUSSION

Security vs. performance trade-offs: In most systems security features come at the cost of degraded performance, and our proposed DDoS countermeasures for CDNs face a similar trade-off. As described in Section IV, the hash-based request routing scheme is unlike a standard CDN request routing algorithm that chooses the optimal server based on network or server load, or network proximity. Rather, we assume that each CDN server within a given region provides roughly equivalent performance for clients assigned to that region. To allow further optimization within a region, the approach could be modified to use weighted hash functions, for instance, where the weights are determined using conventional request routing metrics. However, if the request routing function exhibits some predictability based on performance-related information, it may increase the vulnerability of CDN servers to attack. Similarly, the site allocation strategy presented in Section V potentially reduces the performance of an individual Web site by assigning it to fewer CDN servers.

CDN server distribution and footprint: The size and distribution of the CDN influences the effectiveness of our DDoS countermeasures. The mechanisms we propose are directly applicable to large CDSPs which currently operate thousands of CDN servers distributed across many networks. On the other hand, if the CDN is composed of a few large regions, each containing a small number of CDN servers, it may be impossible to find a site allocation that provides sufficient Web site isolation. Similarly, the additional

TABLE I

EXAMPLE SITE ALLOCATION FOR 100 WEB SITES WITH DEGREE OF ISOLATION = 3 ($m = 100$ AND $d = 6$)

	# Total server	# Servers/site	Code
Algorithm 1	$n = 16$	$h = 8$	specialized non linear code [22]
Algorithm 2	$n = 21$	$h = 11$	Reed-Solomon code
Algorithm 3	$n = 30$	$h = 15$	Hamming code

TABLE II

SUMMARY OF THE SITE ALLOCATION STRATEGIES USING CODES

Code	Properties	Comments
Hadamard code [20], [21]	$m = 2n - 2$, $h = \frac{n}{2}$, $d = \frac{n}{2}$	good isolation, small number of sites (m)
Reed-Solomon code [20]	$m = O(2^{c_1 n})$, $h = \frac{n}{2}$, $d = \frac{n}{c_2 \log n}$	balances isolation and number of sites, efficient construction
Justesen code [20]	$m = O(2^{c_1 n})$, $h = \frac{n}{2}$, $d = c_2 n$	good isolation, many sites, higher complexity

protection afforded by the hash-based request routing is significantly reduced with a small number of servers. The applicability to small CDNs may be improved, however, with the emergence of CDN peering in which multiple, administratively separate CDNs are combined to create a larger virtual CDN with increased reach and distribution.

VII. CONCLUSION

Recent work on countering DDoS attacks typically has focused primarily on attacks targeting a centralized server location or network resources. Increasingly, however, high-profile sites are distributed using CDNs. While CDNs, owing to their distributed structure, promise better resilience to DDoS attacks, the shared nature of the CDN infrastructure introduce unique challenges. In this paper, we proposed two mechanisms to significantly improve the resilience of CDN-hosted Web sites and CDN servers to DDoS attacks: (a) a hash-based request routing scheme that enables CDN servers to effectively distinguish attack traffic from legitimate requests, and (b) site allocation algorithms, based on coding theory, which guarantee a minimum level of availability of the sites that are not directly under attack. Together, these schemes improve the resilience of CDN-hosted Web sites, and complement existing techniques used to counter DDoS attacks. Several issues remain to be addressed in future work. For example, mechanisms are still needed to secure request routers from attack. Also, a CDSP may wish to support multiple levels of service, or to handle cases where some sites require more or fewer servers. Although the direct relation to codes is not valid, we are developing allocation strategies for multiple classes of service using codes.

REFERENCES

- [1] Kihong Park and Heejo Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," in *Proceedings of ACM SIGCOMM*, August 2001.
- [2] F. Kargl, J. Maier, and M. Weber, "Protecting Web servers from distributed denial of service attacks," vol. 10, May 2001.
- [3] D. J. Bernstein, "SYN cookies." <http://cr.yip.to/syncookies.html>, November 2001.
- [4] David Moore, Geoffrey M. Voelker, and Stefan Savage, "Inferring Internet Denial-of-Service Attack," in *Proceedings of USENIX Security Symposium*, August 2001.
- [5] "Strategies to protect against distributed denial of service (DDoS) attacks." Cisco Systems White Paper, February 2000. <http://www.cisco.com/warp/public/707/newsflash.html>.
- [6] H. Burch and B. Cheswick, "Tracing anonymous packets to their approximate source," in *Proceedings of USENIX Systems Administration Conference (LISA)*, December 2000.
- [7] G. Sager, "Security fun with OCxmon and cflowd." Presentation to Internet-2 Measurement Working Group, November 1998. <http://www.caida.org/projects/ngi/content/security/1198/>.
- [8] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer, "Hash-based IP Traceback," in *Proceedings of ACM SIGCOMM*, August 2001.
- [9] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson, "Practical Network Support for IP Traceback," in *Proceedings of ACM SIGCOMM*, August 2000.
- [10] S. M. Bellovin, M. D. Leech, and T. Taylor. Internet draft (draft-ietf-itrace-01.txt), April 2002.
- [11] CERT Coordination Center, "TCP SYN flooding and IP spoofing attacks." CERT Advisory CA-1996-21, September 1996. <http://www.cert.org/advisories/CA-1996-21.html>.
- [12] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites," in *Proceedings of 11th WWW Conference*, May 2002.
- [13] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of content distribution networks," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2001.
- [14] IBM e-business Hosting, July 2002. <http://www.ibm.com/services/webhosting>.
- [15] A. Barbir *et al.*, "Known CDN request-routing mechanisms." Internet Draft (draft-ietf-cdi-known-request-routing-00.txt), February 2002.
- [16] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of DNS-based server selection," in *Proceedings of IEEE INFOCOM*, April 2001.
- [17] CERT Coordination Center, "Denial of service attacks." CERT Tech Tips, June 2001. <http://www.cert.org/tech.tips/denial-of-service.html>.
- [18] Mihir Bellare and Ran Canetti and Hugo Krawczyk, "Keyed Hash Functions for Message Authentication," in *Proceedings of CRYPTO*, August 1996.
- [19] D. Mosberger and T. Jin, "httpperf: A tool for measuring web server performance," *ACM Performance Evaluation Review*, vol. 26, December 1998.
- [20] R. E. Blahut, *Theory and practice of Error-Control Codes*. Addison Wesley, 1983.
- [21] V. Pless and W. Huffman, eds., *Handbook of Coding Theory*. Elsevier, Amsterdam, 1998.
- [22] M. Best, A. Brouwer, F. MacWilliams, A. Odlyzko, and N. Sloane, "Bounds for Binary Codes of Length less than 25," in *IEEE Transactions on Information Theory*, January 1978.